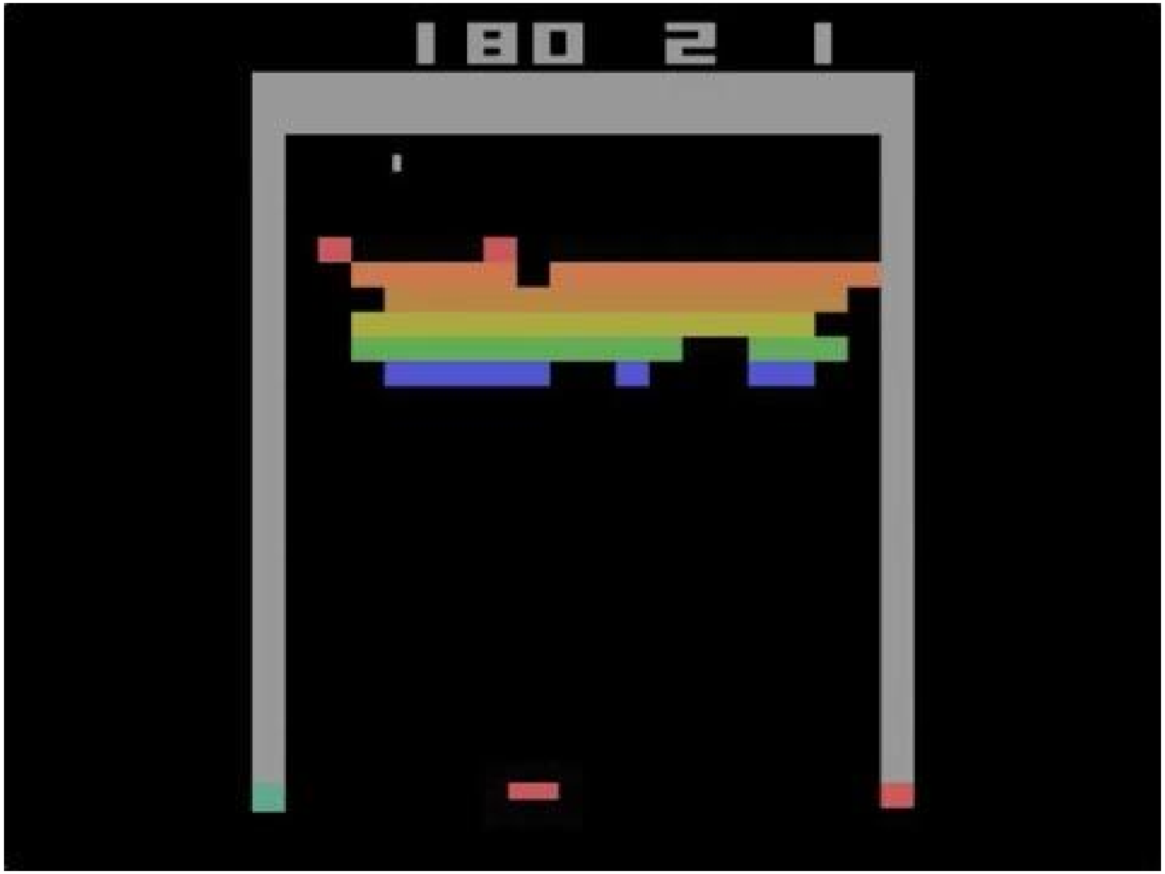
A Practical Introduction to Deep Reinforcement Learning

Chris Lu

Hosted by Machine Learning @ Berkeley

What is Reinforcement Learning?

Video Example: DQN

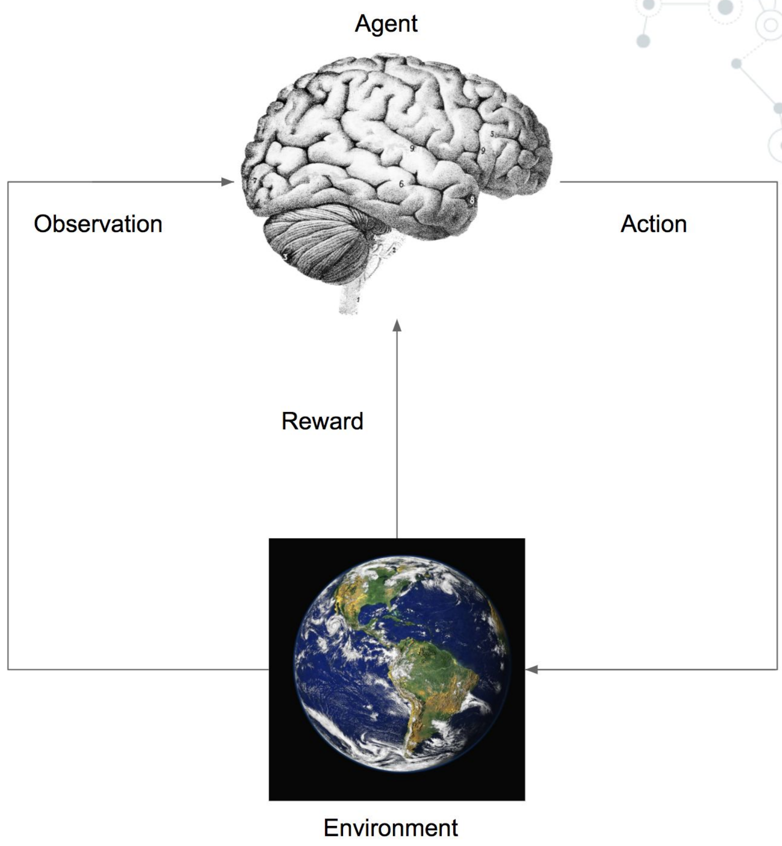


Problem Setup

-At each time step, the agent receives an observation (or state) and a reward. The agent then decides an action and receives the next observation and reward.

-Goal: We want to learn how to act optimally in this general environment.

-Example: In the Atari example, the observation was the pixels of the image, the reward was the score, and the action was a controller input.



Cumulative Reward and Q-Values

-Our goal is to find a policy (a function from state to action) that maximizes expected future reward.

-We also want to prefer immediate reward over delayed rewards. To do this, we define a “Value” to states that represents cumulative discounted reward.

$$V^{p}(s) = \sum_{k=1}^{+\infty} \gamma^{k-1} r_k = r_1 + \gamma r_2 + \gamma^2 r_3 + \dots$$

- $\boldsymbol{\gamma}$ is a hyperparameter between 0 and 1 and is called the “discount”.

-$\mathbf{r_t}$ is the reward at timestep **t**

-The Value Function maps a value to a state, but it does not give us a policy.

-Solution: Estimate Q-Values instead! *Q(s,a)* returns the value of taking an action *a* in a state *s*.

-Now, it is easy to determine what action to take given a state.

$$\underset{a}{\operatorname{argmax}} \{Q(s,a)\}$$

Q-Learning

Initialise the Q' table with random values.

1. Choose an action a to perform in the current state, s.
2. Perform a and receive reward $\mathcal{R}(s, a)$.
3. Observe the new state, $\mathcal{S}(s, a)$.
4. Update:

$$Q'(s,a) \leftarrow \mathcal{R}(s,a) + \gamma \max_{\alpha} \{Q'(\mathcal{S}(s,a), \alpha)\}$$

5. If the next state is not terminal, go back to step 1.

-R(s,a) returns the reward of taking action *a* in state *s*

-S(s,a) returns the next state, *s'*, after taking action *a* in state *s*.

Exploration vs. Exploitation

-How do we pick proper actions?

-Exploring means we will get less reward, but will understand parts of the environment better.

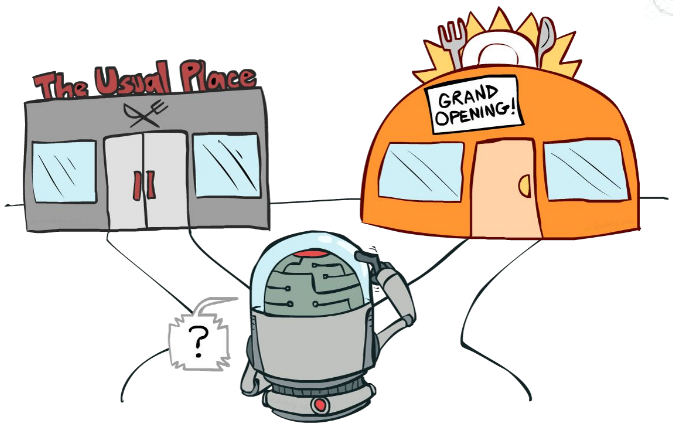
-Exploiting will give us more reward, but we might miss out on discovering a better policy.

-Ideally, we would want to explore more at first and then over time, exploit more.

-One very simple way to do this is called ε-greedy action selection.

-We take a random action with probability ε and decrease ε over time.

-Note that there are many other ways to explore.



Shortcomings of Basic Q-Learning

-Before, we would do Q-Learning by storing the Q-Values in a table for each state-action pair.

-It does not work with continuous state spaces.

-It performs very poorly in very large state spaces.

-It requires a huge table as well for large state spaces.

-For example, for the DQN Atari setup, tabular Q-Learning would have to store:

(84*84*4)*18 = 508032 Values

-What is a solution?

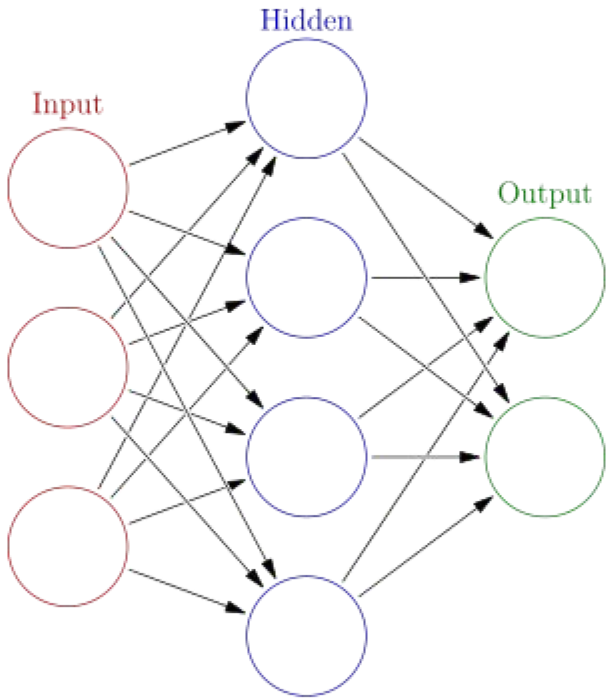
Neural Networks!

On Neural Networks

-This presentation will not go into depth on how neural networks work.

-For the purposes of this workshop, you can think of neural networks as something that approximates a function given a sufficient number of inputs and outputs.

-In this case, we are trying to approximate the function Q(s,a).



Experience Replay

-Neural Networks used to not work well for Reinforcement Learning until experience replay was used for it!

-Training the neural network at each time step with the last action correlates the training of the neural network, which can lead to an unstable network.

-Thus, at each time step, we have to store the state, action, reward, and next state in a buffer and then randomly sample to train the network.

-In the DQN paper (https://www.nature.com/articles/nature14236), the authors relate it to how humans and animals can recall previous experiences to train themselves.

Coding It All Up

Other Methods

-We just went over Deep Q-Learning, but there are some other ways to approach the Reinforcement Learning problem.

-In Policy-Based methods, the neural network attempts to learn a policy (a function from state to action) directly rather than Q-Values. (Sometimes, it does both in Actor-Critic frameworks). It directly updates the policy based on the reward it receives.

DDPG:

https://arxiv.org/abs/1509.02971

A3C:

https://arxiv.org/abs/1602.01783

PPO:

https://arxiv.org/abs/1707.06347

Where to go from here?

-There are some very simple and effective improvements on the code that we just wrote.

-Here are some papers you can read and try to implement. They should be pretty implementable using the code we just wrote!

-This is a large part of what is really cool about Deep Reinforcement Learning: A lot of the ideas are very easily digestible and can be very quickly implemented.

Human-level control through deep reinforcement learning

https://www.nature.com/articles/nature14236

-Published in nature on July 2014, this paper is what led to Deep Reinforcement Learning’s rise in popularity.

-In it, they introduce the DQN algorithm.

-We implemented most of DQN, but they add something called a “target network” that decorrelates the inputs of the neural network even more and prevent instability.

-The “target network” is like an old snapshot of the Q-Network, or “online network”.

Deep Reinforcement Learning with Double Q-learning

https://arxiv.org/abs/1509.06461

-Published in September 2015

-They introduce a small, but important improvement to DQN. They use the target network to estimate the q-values, but they use the online network to take actions.

-In DQN, they used the online network to take actions and estimate Q-Values, but this led to a network that would overestimate the value of states.

Dueling Network Architectures for Deep Reinforcement Learning

https://arxiv.org/abs/1511.06581

-Published in November 2015

-They estimate the average value of a state and the change in that value with each action. This way, it can more easily distinguish the proper action to take in a situation where the overall state is generally good.

Parameter Space Noise for Exploration

https://arxiv.org/abs/1706.01905

-Published in June of this year!

-They find that injecting noise into the parameters/weights of the neural network leads to much better exploration than ε-greedy (which was explained in a previous slide above).

Distributed Prioritized Experience Replay

https://openreview.net/pdf?id=H1Dy---0Z

-Currently under review for ICLR, so we do not know who the authors are.

-They have a large number of threads running the game at the same time to update the experience buffer.

-They use something called “prioritized experience replay” which samples from the experience buffer based on how much it “learns” from the sample.

-There are tons of other papers out there and implementing them can be pretty fun!